# NEH/DFG Bilateral
# Digital Humanities Program

Tools & Concepts for Safeguarding & Researching
Born-Digital Culture

Award Number: HG-229308-15

## Whitepaper

September 22, 2017

Project Directors:

Dragan Espenschied (Rhizome)
dragan.espenschied@rhizome.org

Klaus Rechert (University of Freiburg)
klaus.rechert@rz.uni-freiburg.de

## Introduction

Born-digital works of art are some of society's most at-risk cultural materials. As interactive software-based art, or online networked art, these works are an important record of our cultural and aesthetic history as a digital society.

Preservation of these items requires to take their ***performatic*** aspects into account, accepting that born-digital artifacts are not like traditional, "self-contained" archive objects. Instead, they are able to perform their objecthood when an array of technical elements explicitly outside the artifact align: in the case of a browser-based artwork, that would be the artifact, a browser, maybe a series of browser plug-ins, an operating system, networking connections, screens, input devices, and so forth.

Without productive abstractions, sound object boundary definitions, and workflows available, memory institutions have not been able to include the performativity of digital objects into their preservation efforts on a structural level.

This 2 year bilateral project, funded by the NEH in the US and the DFG in Germany, brought together 3 memory institutions—Yale University Library, the German Literature Archive in Marbach, the Flusser Archive in Berlin—under the lead of Rhizome in New York and the University of Freiburg (Germany) to find a remedy this situation.

The goal was to enable complex digital objects to be preserved, exchanged and managed, published, referenced and reviewed.

The research  has been carried out on artworks, with exacting demands on representation and performance. Yet the presented findings are applicable for any type of digital object. Actionable findings have been implemented as new workflows and features into the emulation framework EaaS.[1]

This document presents the abbreviated findings of the research.

## Defining and Maintaining Born-Digital Object Boundaries

The term ***digital object*** is used for a wide range of artifacts, ranging from individual files to imaged media (like a CD-ROM "iso" file) or even full computer systems, including hardware. "Object" is also used interchangeably with the "content" or "role" of an artifact. For instance, a CD-ROM is not thought of as the physical disk, but as the things that happen to a computer when the CD-ROM would be inserted into the drive; a Word document is not about the bitstream, but about the simulacra of a document that appears on screen when it is loaded into Microsoft Office; and of course an executable program or app is not about the bitstream, but about its performance.

[1] See http://emulation.solutions

In order to develop strategies and tools for such a variety of items and roles, it is necessary to systemize the object's technical structure, in particular decoupling it conceptually and technically from a highly specific setup and place it into a documented, controlled and well understood technical environment, in order to separate concerns about the object's role from its technical performance. This can be achieved with a structural decomposition of an object's runtime environment. Fig. 1 shows a typical technical stack of an digital object, its major components and technical interdependencies.

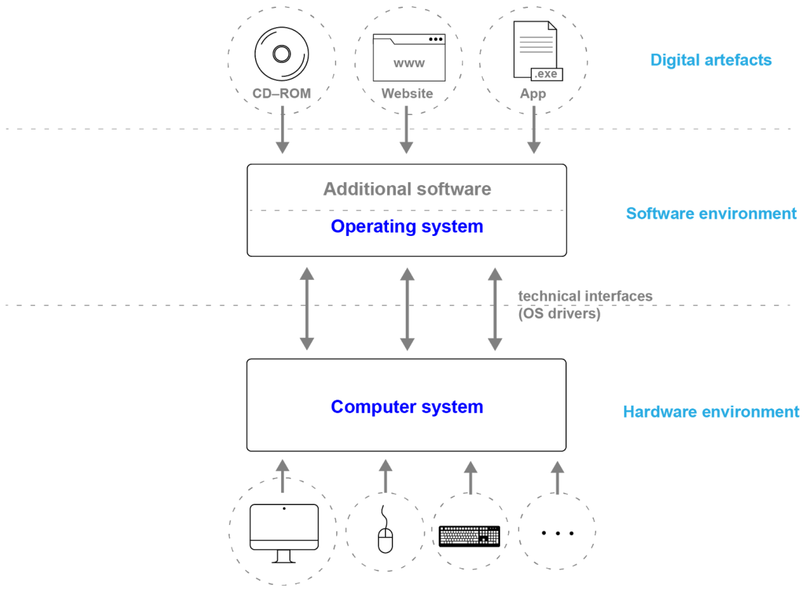


*Fig. 1 : Structural decomposition of a born-digital object*

This technical decomposition results in a first set of conceptual and technical layers which allow to evaluate preservation risk factors and form strategies without considering the specificities of all layers:

- Digital objects are located on top of the technical stack (the **object layer**) which represents the digital object and formulates technical dependencies regarding a software and hardware environment layer.

- The **software environment layer** describes a set of installed software applications, drivers and an operating system, typically available as a virtual disk image, which is usable with emulated hardware. Effectively, the software environment provides an object's performance environment and provides ways for software to connect with the physical world via hardware. Usually, the dependencies of a software environment include the hardware architecture (ISA[2]) and some hardware related capabilities, e.g. network, sound, etc. as well as external interfaces (e.g. USB, serial port).

[2] https://en.wikipedia.org/wiki/Instruction_set_architecture

- The **hardware layer** represents physical or emulated hardware components, connecting a digital object with the physical world.

All hardware will inevitably suffer from physical decay, and for cost reasons can only be preserved in individual cases, for a limited time. This is also true for emulated hardware: as computer systems available in the physical world change and diversify, for instance with ARM instead of Intel becoming the dominant all-purpose computing architecture, the widespread introduction of new input devices such as touch screens or spatial sensors, and new output devices like non-rectangular displays, emulators as a whole will inevitably need to adapt.

In contrast, a software environment's hardware requirements, under preservation care, don't change over time and can be considered stable. **The digital objects performing inside the environment inherit the environment's stability.** Hence, longevity of digital objects can only be ensured if their software environments—the main intermediaries between artifacts and hardware—are maintained over time, i.e. adapted to newly available (emulated) hardware, putting new focus on the preservation of environments.

## Technical generalization of software environments—a crucial step to ensure interoperability and longevity

Given the central role of software environments for preservation of performance, a main requirement for this project was the ability to use real-world disk images with emulation, which can come from two sources:

- disk images originating from physical hardware, such as the artwork ***Bomb Iraq*** by Cory Arcangel and

- disk images that have been created natively within an emulator, with, for instance, a stock install of Windows XP.

**Conceptually these are equivalent**, as both are constructed with dependency on certain hardware, which might be physically present or emulated.

From today's perspective, we can safely assume that there will be a future demand for emulators, e.g. to emulate Intel-based PCs. However, it is impossible to predict their actual technical characteristics: While all emulators of a given technical platform (like the Intel-based x86 platform) support a common ***instruction set architecture*** (ISA), they may differ significantly in their technical configuration, resulting in environments that performed well on a known emulator previously to refuse operation. Hence, it is necessary to prepare conceptually and technically for the event of replacing obsolete emulators with a future generation of emulators.

Within an emulated environment, the operating system (OS) plays an important role, as it typically provides a hardware abstraction. For instance, any application is able to connect to the internet, draw pixels on screen and receive mouse coordinates without knowing technical details about the hardware used. This abstraction is usually achieved through ***technical interfaces***—the so-called

***hardware abstraction layer***. This approach is implemented so widely—in Windows, Linux, Mac OS, Android, etc.—because it greatly simplifies software development and provides compatibility for software with a wide spectrum of hardware.

The abstraction layer is implemented as hardware drivers. Through the use of OS hardware abstraction, any **software, in our case preserved artifacts, does not depend directly on physical hardware components but present *abstracted* hardware dependencies** (such as the minimally required screen resolution, the ability to replay sound, network support, etc.)

In order to find the most effective process, a structured experiment was set up to simulate the migration of environments from one hardware platform to another, based on the currently popular emulation and virtualization packages VMWare, VirtualBox and QEMU, covering the Intel x86 architecture. We installed several popular stock operating system versions of Windows and Linux on all three emulators, using defaults wherever possible, to create typical VMware / VirtualBox / QEMU disk images. Next, we tried to run all of these images on the alternative two emulators.

For brevity, only the work on Windows XP (32bit/SP3) is discussed in greater detail.

| | T1 | T2 | T3 | T1 | T2 | T3 | T1 | T2 | T3 |
|---|---|---|---|---|---|---|---|---|---|
| From \ To | **VMWare** | | | **Virtual Box** | | | **QEMU** | | |
| **VMware** | | | | **-** | **-** | **+** | **-** | **-** | **+** |
| **VirtualBox** | **-** | **+** | **+** | | | | **-** | **-** | **+** |
| **QEMU** | **-** | **-** | **+** | **-** | **+** | **+** | | | |

*Table 1: Migrating a Windows XP disk image from one emulator to another*
**+** indicates success, **-** indicates failure.

**T1: unmodified disk images.** The “naive” approach, using unmodified disk images and emulator defaults, failed for every possible case, usually with “blue screens” caused by mismatched storage drivers, preventing the start of the OS.

**T2: adapting emulator settings.** Re-configuring the emulators’ hardware setup to be more similar to each disk images’ original hardware environment. For all combinations we have searched relevant knowledge bases and community forums for hints or hacks to get Windows into a rudimentary ***safe-mode*** at least, from which repair mechanisms of the operating system could be accessed. Even with significant effort, we have succeeded only in two out of six attempts (cf. Table 1).

**T3: adapting disk images / OS settings.** Since the storage controller was preventing system startup, we evaluated all potential driver configurations for all emulators and were able to select a minimal, technically basic configuration, which should also be compatible with most of the real-world setups. This approach worked every time.

In case of Windows XP,  a script was implemented to modify the environment's storage controller configuration  by modifying the Windows Registry directly on the disk image.

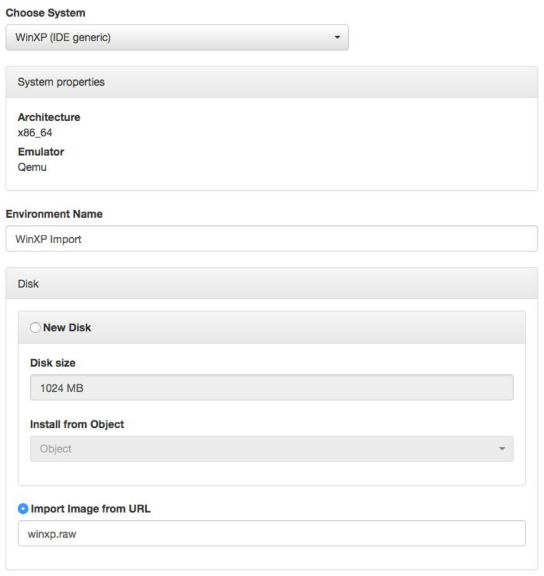
For most real-word disk images as well as for XP stock installations, changing the storage controller is only the first step to a fully functional system. Further necessary adaptations can be done with the help of the operating system itself (e.g. automatic hardware detection and driver installation). In a final step, customization may be performed to improve usability and/or execution speed of the environment.

## Implementation of a generalization and customization process that can be monitored, controlled, and guided through peer-review

These aforementioned results highlight the difficulties of importing environments as disk images from different or unknown hardware configurations, but more importantly they help pointing towards difficulties future changes in emulator setups may cause for archived disk images. Since these migration tasks have to be repeated for every environment, automation is essential. Additionally, the  knowledge created about required changes, manifested as meta-data and tools, will reduce the complexity and cost of future adaptations, especially if **one *generalized configuration* can be applied to all archived disk images featuring the same operating system.**

Based on these insights, an "import + generalization" workflow was implemented. *Generic machine templates* are provided, allowing the user to choose from typical legacy computer systems. These templates describe a systems' technical configuration, but also include automated *generalization procedures*.

A generalization procedure looks out for technical preconditions expected to be present on the disk image: properties such as volume label, system ID, and sets of files and directories. In the case of Windows XP, the bootable partition is identified by looking for Windows system directories. If all preconditions are met, the target partition is made available to the generalization procedures to carry out the configuration adaption. (Fig. 2 shows the corresponding user interface.)



**Choose System**

WinXP (IDE generic)

System properties

**Architecture**
x86_64

**Emulator**
Qemu

**Environment Name**

WinXP Import

Disk

○ **New Disk**

**Disk size**

1024 MB

**Install from Object**

Object

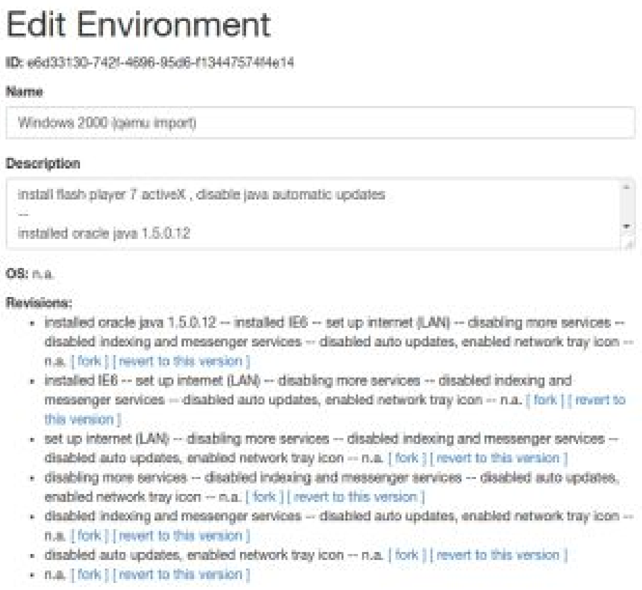
● **Import Image from URL**

winxp.raw

*Fig. 2: EaaS import image dialog*

The first boot of an imported Windows XP image will trigger further automated configuration processes provided by the operating system, e.g. Windows will detect new hardware and may require a few boot-cycles until the system is fully adapted to its new technical environment.

Every disk image is exposed ***read-only*** to emulators and generalization procedures, with a separate writeable layer inserted transparently on top. Any changes to the disk image will be caught in the writable layer while the main disk image is not modified. When a generalization or emulation session has concluded, the writeable layer can be stored as a new ***revision*** of the original image. After importing, at least three revisions will be generated: 1) the original image (which bluescreens), 2) the modifications carried out by generalization procedures, and 3) further adaptations carried out by the operating system's repair functions or the user.



Edit Environment

**ID:** e6d33130-742f-4696-95d6-f13447574f4e14

**Name**

Windows 2000 (qemu import)

**Description**

install flash player 7 activeX , disable java automatic updates
--
installed oracle java 1.5.0.12

**OS:** n.a.

**Revisions:**

- installed oracle java 1.5.0.12 -- installed IE6 -- set up internet (LAN) -- disabling more services -- disabled indexing and messenger services -- disabled auto updates, enabled network tray icon -- n.a. [ fork ] [ revert to this version ]
- installed IE6 -- set up internet (LAN) -- disabling more services -- disabled indexing and messenger services -- disabled auto updates, enabled network tray icon -- n.a. [ fork ] [ revert to this version ]
- set up internet (LAN) -- disabling more services -- disabled indexing and messenger services -- disabled auto updates, enabled network tray icon -- n.a. [ fork ] [ revert to this version ]
- disabling more services -- disabled indexing and messenger services -- disabled auto updates, enabled network tray icon -- n.a. [ fork ] [ revert to this version ]
- disabled indexing and messenger services -- disabled auto updates, enabled network tray icon -- n.a. [ fork ] [ revert to this version ]
- disabled auto updates, enabled network tray icon -- n.a. [ fork ] [ revert to this version ]
- n.a. [ fork ] [ revert to this version ]

*Fig. 3: EaaS user interface for revision handling*

Each of the revisions can be ***forked***, i.e. new logical branches of revisions can be created. All branches are represented as individual environments which are based on the same root disk image and a set of stacked additional layers. Each revision can be annotated to describe the actions that have been carried out, can be started to be examined in action, and connected with an artifact. Similar to a version control system, environments can be reverted to any previous revision if desired.

This workflow can also be used to manually change settings or add software to an environment, providing a full chain of proof to any modifications made to a given environment, as *performatic provenance*.

## Tools and Concepts for working with born-digital objects

By defining a boundary between object and software environment, we decouple object from hardware layer and restrict the technical dependencies to software requirements. The goal is to identify *abstracted dependencies*, i.e. dependencies that are not tied to a computer setup, but only require a certain to be software installed or abstract software interfaces to hardware components (e.g. GPU or external hardware such USB/serial, etc.). **Instead on relying on a mixture of software- and hardware requirements, an object boundary defined by abstracted dependencies makes the object rely only on software and software interfaces.**

This way objects are usable with multiple, different environments, which increases both their resilience due to diversified preservation and presentation options.

Based on these observations and requirements we have implemented tools and workflows to support the object boundaries concept. **All workflows have been implemented in a way to discourage mixing of object and software environment.** Instead, we improved the workflows to create *object-environments*—tailored software environments derived from general environments—to publish or present an object with a specific runtime context. The coupling of artifact and environment is implemented only on a metadata level, where an artifact refers to a specific environment layer, relying on the emulation framework to bring both components together on demand.

This way, an artifact can be kept unmodified in its dedicated repository.The framework implements necessary media conversions to connect an artifact with an emulator. For instance, if an artifact consists of a set of files and directories, the framework will wrap these files into an appropriate media type (e.g. CD-ROM or floppy disk image), depending on characteristics of emulator and software environment.

## Extending the Scope of Emulation

In the last project phase, we were able to work on additional use-cases through a collaboration with TATE Modern in context of the PERICLES EU project, the artist duo Tale of Tales (creators of an OpenGL 3D computer game), and a collaboration with the University of Amsterdam and LIMA. These collaborations gave us access to objects with complex system requirements, e.g. artworks distributed over multiple machines, specific hardware dependencies (like high-performance GPUs) or peripherals, such as cameras and sensors. We verified the proposed methodology on these artworks and tried to identify further technical requirements for improving the EaaS framework.

An important subset of hardware-related dependencies are externally connected hardware components. Following the boundaries concept, we focused on the abstracted communication between an object and external components, especially on protocols and software interfaces used (i.e. how information is exchanged between software and external hardware). Fortunately, the amount of types of external technical interfaces is low and built on general purpose standards (such as USB, serial, parallel etc.)

To connect external hardware components, a physical machine is required. In the case of an emulated computer system, the host system running the emulator needs to be able to connect and interact with external hardware components such as human interface devices (e.g. mouse and keyboard), printers or other peripherals, by using a suitable connector to provide a compatible connection. The host operating system then needs to provide a software interface for applications to communicate with external hardware, an emulator (acting as a normal software application) is then able to use this external hardware. Finally, the emulator needs to provide a virtual hardware interface connected to the host's software interface, visible to and usable by the guest environment. Through all these layers, the integrity of data protocols needs to be maintained.

The artwork ***The Graveyard*** (2008) by Tale of Tales[3] exists in a version for Linux and controls the GPU via the cross-platform OpenGL abstraction layer. Linux already supports a translation from OpenGL calls from inside an emulator to the host system's OpenGL interface via the simulated Virgil3D graphics card.[4] Using this translator, it was possible to run the artwork on an EaaS node in a cloud setup including a GPU, with a high quality frame rate.

[3] http://tale-of-tales.com/TheGraveyard/
[4] https://virgil3d.github.io/

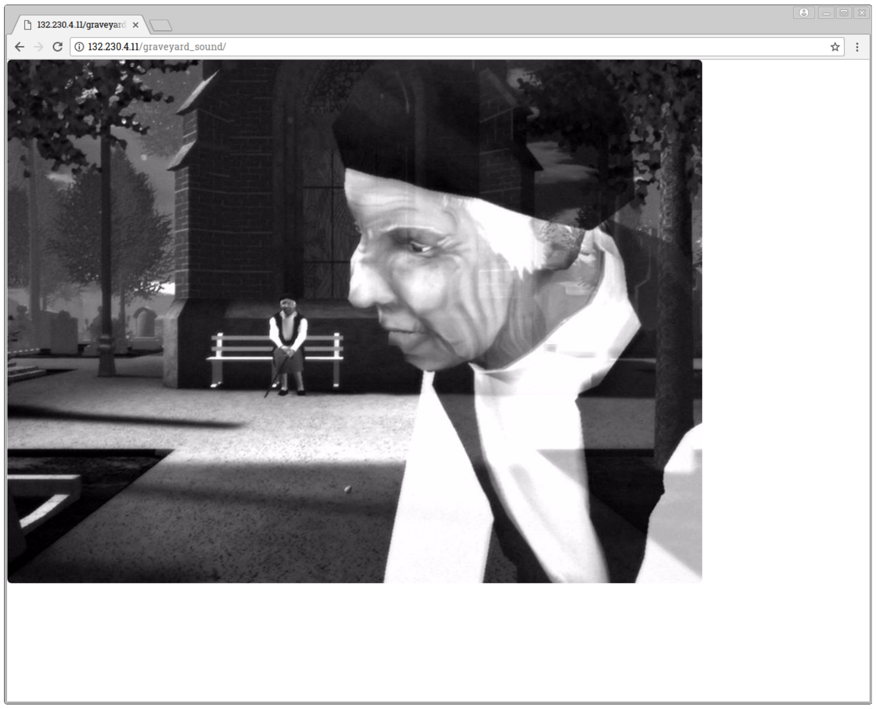


*Fig. 6: The Graveyard running in an experimental setup on a GPU-equipped cloud computing service.*

In a similar case from TATE's collection, an artwork is based on GPU-accelerated 3D rendering, this time controlled via the Windows 3D abstraction layer DirectX. It is likely that a suitable substitute will become available and technically feasible. For another piece created on Windows that runs distributed on a local network and produces images based on the input of Firewire cameras, abstracted networking and abstracted camera input (USB instead of Firewire) was routed into the emulation environment.

**None of the identified technical issues (with regard to emulation) are object-specific**: network connections between Windows computers and connecting GPUs or web cameras are generic problems.

Based on this observation, we extended the EaaS framework to incorporate additional emulation layers: Infrastructure has been built to "inject" a USB data stream either from a remote machine or synthetically created by software. Furthermore, we added capabilities to use GPUs within emulators as well as using GPUs for streaming an emulator's visual output in high resolution and high frame rates.

Comparing the behaviour of an original and an emulated version of a software can reveal a common base-line of abstraction for artifacts within a collection. Any information that goes beyond this technical foundation might be revealed by more diversified technical probing, or requires cultural knowledge about the applied systems. This cultural knowledge is not observable or implicitly present in digital artifacts themselves, but needs to be supplied from other sources.

For instance, a contemporary emulator produces feedback that can be understood in the moment it is used—error messages will be shown in a way that integrates within current conventions of computer usage. The Windows XP running inside the emulator might provide feedback in ways that are not easily recognizable or understandable at the point in time the re-enactment happens.

Finally, a qualitative evaluation of an object's performance inside a software environment can not be based on technical processes alone. The more complex behaviours an object exposes, the more likely performatic variability might be introduced that changes the perception of the object or even challenges its objecthood. Only knowledge about the objects themselves can guide an evaluation process, and might need to rely on documentation of the object managed independent from the emulation framework. Especially in the case of artworks, curators have to decide if a presentation via emulation makes sense. This does however not affect the benefits gained from boundary definition and generalization: even if an object cannot be performed to full satisfaction at one point in time, as emulators and emulation frameworks are developed further, the performance can improve in the future. Additionally, introducing technical and conceptual separations via the suggested processes will be of benefit if any other strategy than emulation is chosen for re-performance.

## Improved Object Citation and Emulation Access

During this project we worked on accessibility, usability and documentation:

The **EaaS Desktop** has been developed to run EaaS as a local application. There are no specific hardware/software requirements beside a working Docker installation (see below) which has become available for all major operating systems.[5] Via a user-friendly interface, curators are able to work with objects locally, e.g. prepare software environments, test objects with various environments, and share or publish the results.

To support sharing, citing or publishing of digital objects, EaaS currently provides multiple options which we have been simplified and improved in the course of this project:

A **JavaScript client** has been developed to support either seamless integration of emulation into an existing Web UIs, or to create individual landing pages e.g. as targets of Handle-URLs or similar PIDs. Within the client JavaScript library, an instance of a running environment is represented as a JavaScript object, a RESTful API connects it with the EaaS framework. This way, publishers are free in adopting the presentation to their needs.

We have also improved the **EaaS-Cloud deployment** options by simplifying installation and maintenance as well as broadening the supported Cloud providers. This enabled Rhizome to run and maintain their own EaaS instance in the Google Cloud and to present multiple artworks to a global audience (see section "Deployment at Rhizome" for details).

5 http://openpreservation.org/blog/2017/09/15/getting-started-with-emulation-the-eaas-desktop-application/

An **EaaS USB appliance** can be used to make an exported environment portable. The improved version of the EaaS USB appliance is now fully customizable by editing simple configuration files. A custom UI to chose from a list of environments stored on the USB drive can be created with only moderate technical skills using standard HTML, based on the EaaS JavaScript client.  Using the same techniques, it is also possible to set up the appliance to boot into an emulated environment directly.

Both the interactive and the direct boot options have been used in public exhibitions at Yale University, Haus der Elektronischen Künste in Basel, MU in Eindhoven, the Whitechapel Gallery in London, the Vancouver Art Gallery, and the DCA in Dundee.

One outcome of achieving this milestone is that Rhizome, with additional support from Google, has established an emulation service on the public Google Compute Cloud, which is planned to be opened to partnering institutions. Yale University is preparing the deployment of an internal emulation service.

## Deployment at Rhizome and use in artistic programming

In the frame of the project and with financial support from Google, the University of Freiburg and Rhizome built an Emulation as a Service infrastructure that has been put to regular use when Rhizome started the online exhibition program Net Art Anthology[6] in 2016. At time of this report, 16 artworks have been published on Rhizome's web site (see Appendix Table A1 for a complete list), fully based on the EaaS workflows described above.

The interaction quality of an environment accessed via the web is highly dependent on the distance of the emulator to the end user, and therefore mirrored in three locations to reach Rhizome's English speaking audiences in the US, Europe and Australia.

In each region, a base set of four EaaS instances is constantly running to serve a base level of user demand. Upon the announcement of a new Net Art Anthology piece via social media, demand is typically spiking, causing the framework to dynamically allocate and manage more virtual machine instances to run emulators.

The establishment of this public EaaS infrastructure has significantly reduced the required restoration work that would have gone into an online presentation of legacy net art. In many cases it was sufficient to reuse a previously prepared environment—for instance one containing the immensely popular browser Netscape 4.8 with the most dominant plugins installed—and to point the browser to a web archive or containerized web server to reproduce the work in question. While the initial customization of such a reusable environment takes some time, any subsequent application doesn't take more than 5 minutes to set up.

All environments have been created, managed and exported using the EaaS desktop application.

[6] See http://anthology.rhizome.org/

Additionally, Rhizome has produced the presentation for an exhibition of ***The Theresa Duncan CD-ROMs*** in Dundee, Scotland, at the DCA, using the exact same environments configurations that are shown online in a museum exhibition via EaaS USB appliances. Earlier in the project, prototypical USB appliances have been used in the exhibitions ***Electronic Superhighway*** (Whitechapel Gallery, London), ***Mashup*** (Vancouver Art Gallery), ***MBCBFTW*** (HeK, Basel and MU, Eindhoven). The appliance made it possible for Rhizome and independent artists to loan legacy work for exhibitions in easy-to-deploy, exhibition-ready setups.[7]

## Conceptual object boundaries, knowledge management and professional roles

### A new method to describe dependencies

The research carried out in this project produced productive technical abstractions for the preservation of performatic digital objects. These abstractions are mirrored in conceptual object boundaries, the management of preservation knowledge at the level of memory institutions and domains, and professional roles in the preservation field.

On the outset, the technical definitions of a digital object (or "digital artifact" when regarding the bitstream storage) have to be equivalent with the forms they are entering institutional care: as single files or sets of them, storage media like CD-ROMs, or whole computer systems (with the built-in storage media representing the artifact.) **In this context, an object is defined as a "unique" digital artifact—unique in the sense that it easily distinguishable from "stock" components like operating systems or standard software tools.**

**We have determined that the traditional way of describing the dependencies that an object requires for its performance is not productive**, because the the timespan information available during ingest would remain actionable is typically very short, the information is too vague to be useful, and ultimately too detailed and not providing meaningful differentiators in between objects.

For example, a CD-ROM published in 2003, containing software for a Windows operating system, would typically list requirements like "VGA graphics, SoundBlaster 16, 320 MB hard disk, USB camera, and 14" color screen"—such components are not available anymore today and describe rather miniscule technical details that the Windows operating system would actually handle in its hardware abstraction layer. Additionally, many additional dependencies might be required but not listed, since they were assumed to be available by default, such as a keyboard and a two-button mouse. A second object might list slightly different requirements, for instance "Firewire Camera." This information might suggest the need for two separate environments supporting different types of cameras, when in fact any type of camera would have done the job, again due to hardware abstraction.

[7] See D Espenschied, O Stobbe, T Liebetraut, K Rechert (2016): Exhibiting Digital Art via Emulation. In Proceedings of the 13th International Conference on Digital Preservation (iPres16)

Therefore, the *abstracted dependency* of both objects is an operational Windows environment that is able to interface with a camera, with the technical details being managed at the environment layer.

This has certain implication for the definition of object boundaries and the management of objects in collections: technical boundaries, and therefore preservation risks, are becoming apparent when the digital object is placed in a preservation environment. **Errors or wrong behavior during performance highlight boundaries at which the emulation framework needs to provide translations or substitutes**, such as translating the video signal of a USB-C camera connected to the host to a USB 1.0 stream in the environment, or providing a transparent IPv6 to IPv4 converter. However, these are not related to the object, but to the environment. Since legacy environments do not change anymore, the emulation framework can over time retrospectively provide all features that any object might ask for.

**Hence, the description of technical dependencies and boundaries only make sense at the environment level, as all objects using a certain environment share this environment's dependencies.** Description of *abstracted demands* (object) and *capabilities* (environment)—like networking, sound—can help to match environments with artifacts.

From the perspective of a curator of objects, the emulation framework represents a single layer that provides usable environments, largely eliminating the need to know about hardware configurations.

## Professional roles

The technical separation described above finds its equivalent a division of labor and roles in preservation:

- The foundation for the preservation of performatic digital objects is the **emulation framework**. It provides a finite set of working base environments, complete with abstractions for interaction with the outside world, like input and display devices, networks, and more. The framework handles changes to base and derivative environments with the goals is to keep each base environment, and in consequence their derivatives, usable. This requires knowledge about interfaces in between emulator and host, emulator and client used for access, and existing hardware abstraction already built into environments. (For instance, the framework might need to provide an on-screen keyboard when a touchscreen device is used to access an environment; the framework needs to release generalization procedures to keep environments usable.) The technical work involved in role is specialized and "expensive," and is to the benefit of a wide array of domains: museum restorators, artists, engineers, etc, who are seeking to preserve their digital materials will base their work on the same base environments.

- On the object facing side of an environment, a **software curator** customizes the base environments to fit the class of objects they need to re-enact. Based on technical knowledge specific to their domain, they collect and describe the capabilities of software tools and other means of customization. For instance, a net art institution would collect different networking tools like browsers, an architecture museum would collect CAD software, an administrative archive would collect versions of office software, and so forth. The work done in this layer can be beneficial across a whole domain if an environment is customized for a common use-case, or benefit just a small range of objects.The technical knowledge required for this role is that of a “power user” or system administrator, without the need to know anything about hardware.

- The **object curator** creates combinations of objects and environments for research, general access, or publication, using matching aids provided by the software curator. The object curator’s knowledge about specific objects allows them to create the right context for accessible versions and evaluate the quality of performances. The technical knowledge required for this role is that of a regular user, their work benefits the audience of a collection.

## Conclusion

This project presents new concepts and workflows to bound, generalize and maintain performatic digital objects, which can be implemented at memory institutions dealing with digital culture. Research results are manifested in the open source emulation framework EaaS.[8] Rhizome has implemented the framework into the workflows of its artistic program and will soon offer emulation services to arts institutions. We’d like to hear from you, send us email! The future will be emulated.

Klaus & Dragan

[8] https://github.com/eaas-framework

## Appendix

| Title | Creator | Year | Access URL | Notes |
|---|---|---|---|---|
| re-move.org | Lia | 1999-2003 | http://archive.rhizome.org/anthology/re-move.html | Generative art work depending on legacy browsers and plugins (Shockwave) |
| skinonskinonskin | Entropy8Zuper | 1999 | http://archive.rhizome.org/anthology/skinonskinonskin.html | Collaborative net art work depending on legacy browsers and plugins (DHTML, Flash, Shockwave, RealAudio) |
| World Of Awe | Yael Kanarek | 2000 | http://archive.rhizome.org/anthology/world-of-awe.html | Net art work relying on legacy browser |
| Airworld | Jennifer & Kevin McCoy | 1999 | http://archive.rhizome.org/anthology/airworld.html | Net art work relying on Quicktime |
| Data Diaries | Cory Arcangel | 2003 | http://archive.rhizome.org/anthology/data-diaries.html | Net art work relying on Quicktime |
| Heritage Gold | Mongrel | 1997 | http://archive.rhizome.org/anthology/heritage-gold.html | Modified version of Photoshop |
| Blacklash | Mongrel | 1998 | http://archive.rhizome.org/anthology/blacklash.html | Modified action game |
| The Web Stalker | I/O/D | 1997 | http://archive.rhizome.org/anthology/webstalker.html | Conceptual web browser |
| I/O/D 3 | I/O/D | 1996 | http://archive.rhizome.org/anthology/iod3.html | File-system specific work circulated on diskette |
| Bodies© INCorporated | Victoria Vesna | 1996-1999 | http://archive.rhizome.org/anthology/bodiesinc.html | VRML (Virtual Reality) net art and installation work |
| The File Room | Antoni Muntadas | 1994-1998 | http://archive.rhizome.org/anthology/thefileroom.html | Emulation used to run the server (ColdFusion scripting language) |
| Bomb Iraq | Cory Arcangel | 1992, 2005 | http://media.rhizome.org/emulating-bomb-iraq-arcangel/index.html | Whole system preservation, migrated to new, multi-region emulation infrastructure |
| Chop Suey | Theresa Duncan | 1995 | http://archive.rhizome.org/theresa-duncan-cdroms/ | CD-ROM games, migrated to new, multi-region emulation infrastructure |
| Smarty | | 1996 | | |
| Zero Zero | | 1997 | | |
| Epithelia | Mariela Yeregui | 1999 | http://archive.rhizome.org/anthology/epithelia.html | Net art work tied to a specific Netscape version |

Table A1: Artworks publicly presented on the web by Rhizome, using EaaS